NCPC 2022
Presentation of solutions

2022-10-08

BAPC 2021 Preliminaries

Solutions presentation

October 9, 2021

DAPC 2022

Solutions presentation

September 30, 2022

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Solution

1. We want to know who has what score. If Bob is serving, swap the scores.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Solution

1. We want to know who has what score. If Bob is serving, swap the scores.
2. The serves follow the pattern *ABBAABBAABBAABBA...*, so Bob is serving when $x + y = 1$ or $x + y = 2$ modulo 4.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Solution

1. We want to know who has what score. If Bob is serving, swap the scores.
2. The serves follow the pattern *ABBAABBAABBAABBA...*, so Bob is serving when $x + y = 1$ or $x + y = 2$ modulo 4.
3. We now know who has what score at each point in the game. There are only two ways a log can be invalid:
    1. A player's score decreases.
    2. The game continues after someone reaches 11. This includes the tricky case $11 - 11$.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?
- **Solution:** It is fast enough to simulate the algorithm and count the number of iterations. Runtime: $\mathcal{O}(n^2)$.

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?
- **Solution:** It is fast enough to simulate the algorithm and count the number of iterations. Runtime: $\mathcal{O}(n^2)$.
- **Optimized:**
  - Observe that high numbers move to the right immediately, and low numbers move $k-1$ to the left per iteration.
  - Solution: find the maximum distance (to the left) of any value to their sorted position (D), and output $\left\lceil \frac{D}{k-1} \right\rceil$.
  - Runtime: $\mathcal{O}(n \log n)$ for sorting, $\mathcal{O}(n)$ for finding the maximum distance.

B: Bubble-bubble Sort

Problem Author: Ragnar Groot Koerkamp

- **Problem:** How many iterations of Bubble-bubble Sort should you run?
- **Solution:** It is fast enough to simulate the algorithm and count the number of iterations. Runtime: $\mathcal{O}(n^2)$.
- **Optimized:**
  - Observe that high numbers move to the right immediately, and low numbers move $k-1$ to the left per iteration.
  - Solution: find the maximum distance (to the left) of any value to their sorted position (D), and output $\left\lceil \frac{D}{k-1} \right\rceil$.
  - Runtime: $\mathcal{O}(n \log n)$ for sorting, $\mathcal{O}(n)$ for finding the maximum distance.

Statistics: 73 submissions, 29 accepted, 27 unknown

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Solution

1. Iterate from left to right, and keep a variable `coffee` that stores how many cups you are holding.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Solution

1. Iterate from left to right, and keep a variable `coffee` that stores how many cups you are holding.
2. When encountering a coffee machine: `answer += 1` and `coffee = 2`.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Solution

1. Iterate from left to right, and keep a variable `coffee` that stores how many cups you are holding.
2. When encountering a coffee machine: `answer += 1` and `coffee = 2`.
3. When entering a no-coffee room, do nothing if `coffee == 0`. Otherwise, `coffee -= 1` and `answer += 1`.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

D: Dickensian Dictionary

Problem Author: Mees de Vries

- **Problem:** Given a word, decide if it is *Dickensian*
  (i.e., typeable alternatingly with left and right hand)

D: Dickensian Dictionary

Problem Author: Mees de Vries

- **Problem:** Given a word, decide if it is *Dickensian*
  (i.e., typeable alternatingly with left and right hand)
- Solution: Check for every letter whether it is typeable with left or right

D: Dickensian Dictionary

Problem Author: Mees de Vries

- **Problem:** Given a word, decide if it is *Dickensian*
  (i.e., typeable alternatingly with left and right hand)
- Solution: Check for every letter whether it is typeable with left or right
- Check if the resulting list is alternating
  - Note that you can start with either left or right

E — Enigmatic Enumeration

Problem

Given an undirected graph, count the number of shortest cycles.

Statistics at 4-hour mark: 59 submissions, 12 accepted, first after 00:58

E — Enigmatic Enumeration

Problem

Given an undirected graph, count the number of shortest cycles.

Solution

1. Step 1: Find k, the length of the shortest cycle. For every vertex, find the length of the shortest cycle containing that vertex. This is the distance from the vertex to itself, and can be found with BFS.

Statistics at 4-hour mark: 59 submissions, 12 accepted, first after 00:58

E — Enigmatic Enumeration

Problem

Given an undirected graph, count the number of shortest cycles.

Solution

1. Step 1: Find k, the length of the shortest cycle. For every vertex, find the length of the shortest cycle containing that vertex. This is the distance from the vertex to itself, and can be found with BFS.
2. Step 2: For every vertex v, count the number of shortest cycles going through v. Do a BFS again:
    1. If k is even, for every vertex w of distance $k/2$ from v, count the number of pairs of shortest paths to it.
    2. If k is odd, count the number of edges (u, w) such that u and w are at distance $\frac{k-1}{2}$ from v.

Statistics at 4-hour mark: 59 submissions, 12 accepted, first after 00:58

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.
2. $s = l_1 + l_2 + \cdots + l_n$ can be obtained by adding up all lengths in the input.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.
2. $s = l_1 + l_2 + \cdots + l_n$ can be obtained by adding up all lengths in the input.
3. $(n-2) \cdot l_i + s$ can be obtained by adding all lengths in row i.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.
2. $s = l_1 + l_2 + \cdots + l_n$ can be obtained by adding up all lengths in the input.
3. $(n-2) \cdot l_i + s$ can be obtained by adding all lengths in row i.
4. If $n > 2$, this gives us all lengths l_i. All strings can now easily be obtained.

F — Foreign Football

$n = 2$

New problem: we have two strings a and b, and want to know if there exist strings s, t such that $a = s + t$ and $b = t + s$.

1. Method 1: Try every way of cutting a into $s + t$, use string hashing to check if $b = t + s$.
2. Method 2: Take $b + b$, and check if a is a substring (KMP or hashing).

Statistics at 4-hour mark: 109 submissions, 13 accepted, first after 01:32

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.
2. Use dynamic programming to calculate the probability to score at least k points when answering the last m questions, for every m.

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.
2. Use dynamic programming to calculate the probability to score at least k points when answering the last m questions, for every m.
3. $dp(m, q) =$ probability that we get exactly q points when answering the m last questions.

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.
2. Use dynamic programming to calculate the probability to score at least k points when answering the last m questions, for every m.
3. $dp(m, q) =$ probability that we get exactly q points when answering the m last questions.
4. $dp(m, q) = p_m \cdot dp(m-1, q-1) + (1-p_m) \cdot dp(m-1, q+1)$.
5. Time complexity: $\mathcal{O}(nk)$.

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \leq h_{i+1} \leq \cdots \leq h_j \geq h_{j+1} \geq \cdots \geq h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \leq h_{i+1} \leq \cdots \leq h_j \geq h_{j+1} \geq \cdots \geq h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

1. For every index i, calculate $L(i)$, the minimum j such that $h_j \leq h_{j+1} \leq \cdots \leq h_i$.

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \leq h_{i+1} \leq \cdots \leq h_j \geq h_{j+1} \geq \cdots \geq h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

1. For every index i, calculate $L(i)$, the minimum j such that $h_j \leq h_{j+1} \leq \cdots \leq h_i$.
2. Similarly, calculate $R(i)$, the maximum j such that $h_i \geq \cdots \geq h_j$.

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \le h_{i+1} \le \cdots \le h_j \ge h_{j+1} \ge \cdots \ge h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

1. For every index i, calculate $L(i)$, the minimum j such that $h_j \le h_{j+1} \le \cdots \le h_i$.
2. Similarly, calculate $R(i)$, the maximum j such that $h_i \ge \cdots \ge h_j$.
3. The height of the hill with peak at i is $\min(h_i - h_{L(i)}, h_i - h_{R(i)})$. Take the maximum of these.

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

icpc.foundation

icpc international collegiate programming contest

ICPC North America Championship 2021

You Be the Judge, Again

The Problem

- Given a $2^n \times 2^n$ array filled with numbers in the range from 0 to $\frac{4^n-1}{3}$, determine if it is correctly filled with exactly $\frac{4^n-1}{3}$ L-shaped triominos, one each of the integers from 1 to $\frac{4^n-1}{3}$, with exactly one 0

Right:

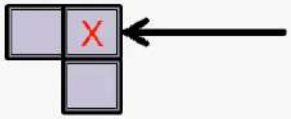
1	1	2	2
1	3	3	2
4	4	3	5
4	0	5	5

Wrong:

1	1
1	1

The Solution

- There's only a couple of things you need to check
  - There's exactly one 0
  - There's exactly three of each of the other $\frac{4^n-1}{3}$ integers
  - There's exactly $\frac{4^n-1}{3}$ "elbows"



That's an "Elbow"

Counts

- There must be exactly one 0 and exactly three of all of the others.
- The problem guarantees that all the numbers are between 0 and $\frac{4^n-1}{3}$
- So, if there are too few of one, there has to be too many of another. Just look for overages.

```
int m = 1<<n;
// Read in the data, make sure there's exactly one 0
// and three of everything else.
// If there's less than three of something,
// there'll be more than three of something else.
for( int i=0; i<m; i++ ) for( int j=0; j<m; j++ )
{
    int x = sc.nextInt();
    tiling[i][j] = x;
    ++counts[x];
    if( counts[x] > (x==0 ? 1 : 3) ) success = false;
}
```

Elbows

```
int k = (1<<(n+n))/3; // That’s (4^n-1)/3

// Now test that the three cells of each tile form an L shape.
// there should be exactly one center, with one cell (up or down), and one cell (left or right)
for( int i=0; i<m; i++ ) for( int j=0; j<m; j++ )
{
    int sum = 0;
    int x = tiling[i][j];

    // Up/Down
    if( i>0 && tiling[i-1][j]==x ) sum += 1;
    if( i<m-1 && tiling[i+1][j]==x ) sum += 1;

    // Left/Right
    if( j>0 && tiling[i][j-1]==x ) sum += 2;
    if( j<m-1 && tiling[i][j+1]==x ) sum += 2;

    // No way to get three unless there's exactly one up/down and exactly one left/right.
    // Since we've already determined that there's exactly 3 of each positive number,
    // this can't happen more than once per triomino.
    if( sum==3 ) --k;
}

// Make sure we got 'k' triominos
if( k>0 ) success = false;
```

J: Jack the Mole

Problem Author: Pim Spelier

- **Problem:** Given n integers between 1 and 1000, find which of them can be removed such that the remainder can be partitioned into two sets of equal sum.

J: Jack the Mole

Problem Author: Pim Spelier

- **Problem:** Given n integers between 1 and 1000, find which of them can be removed such that the remainder can be partitioned into two sets of equal sum.
- Define $s := n \cdot w$ to be the sum of the integers.

J: Jack the Mole

Problem Author: Pim Spelier

- **Problem:** Given n integers between 1 and 1000, find which of them can be removed such that the remainder can be partitioned into two sets of equal sum.
- Define $s := n \cdot w$ to be the sum of the integers.
- $\mathcal{O}(n^2 \cdot s) = \mathcal{O}(n^3 \cdot w)$ solution: For each mole run a $\mathcal{O}(n \cdot s)$ knapsack to check if a partitioning is possible.
  This is usually too slow, unless using bitsets in C++.

J: Jack the Mole

Problem Author: Pim Spelier

- **Problem:** Given n integers between 1 and 1000, find which of them can be removed such that the remainder can be partitioned into two sets of equal sum.
- Define $s := n \cdot w$ to be the sum of the integers.
- $\mathcal{O}(n^2 \cdot s) = \mathcal{O}(n^3 \cdot w)$ solution: For each mole run a $\mathcal{O}(n \cdot s)$ knapsack to check if a partitioning is possible.
  This is usually too slow, unless using bitsets in C++.
- $\mathcal{O}(n^2 \cdot w)$ solution:
  - For each prefix of moles, compute all possible weights of a subset in $\mathcal{O}(n \cdot s)$.
  - For each suffix of moles, compute all possible weights of a subset in $\mathcal{O}(n \cdot s)$.
  - Mole i can be left out if it is possible to make a subset of size l with the moles before i, and a subset of size $(s - w_i)/2 - l$ of the moles after i, for some l.

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

Solution

1. How to check if two substrings are equal? String hashing!

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

Solution

1. How to check if two substrings are equal? String hashing!
2. Use a segment tree to store the hashes, so that it can handle updates.

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

Solution

1. How to check if two substrings are equal? String hashing!
2. Use a segment tree to store the hashes, so that it can handle updates.
3. When we get a palindrome query, we will learn that some characters are identical.

K — Keyboard Queries

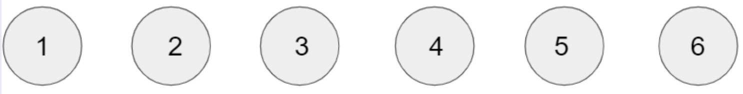
Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

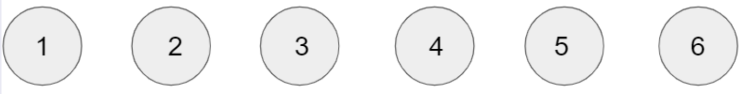
Solution

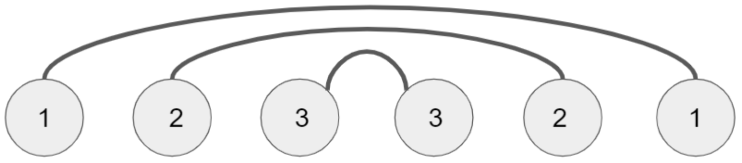
1. How to check if two substrings are equal? String hashing!
2. Use a segment tree to store the hashes, so that it can handle updates.
3. When we get a palindrome query, we will learn that some characters are identical.
4. Use union-find to keep connected components of identical characters. When merging, take the smaller component and change all of its characters.

K — Keyboard Queries



K — Keyboard Queries





K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!
3. There are only $\leq n-1$ merges in total.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!
3. There are only $\leq n - 1$ merges in total.
4. Another optimization: When getting a query of type 2, start by checking if it is already a palindrome.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!
3. There are only $\leq n - 1$ merges in total.
4. Another optimization: When getting a query of type 2, start by checking if it is already a palindrome.
5. Time complexity: $\mathcal{O}(n \log^2 n + q \log n)$.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

TraveLog Solution Slides

North American Championships 2021

Arnav Sastry

August 14, 2021

Formal Statement

You are given a directed graph G and a "TraveLog" of times. Count the number of paths from vertex 1 to vertex n such that the set of shortest times from city 1 to the cities on the path is a subset of the given TraveLog. If there is exactly one path, output it.

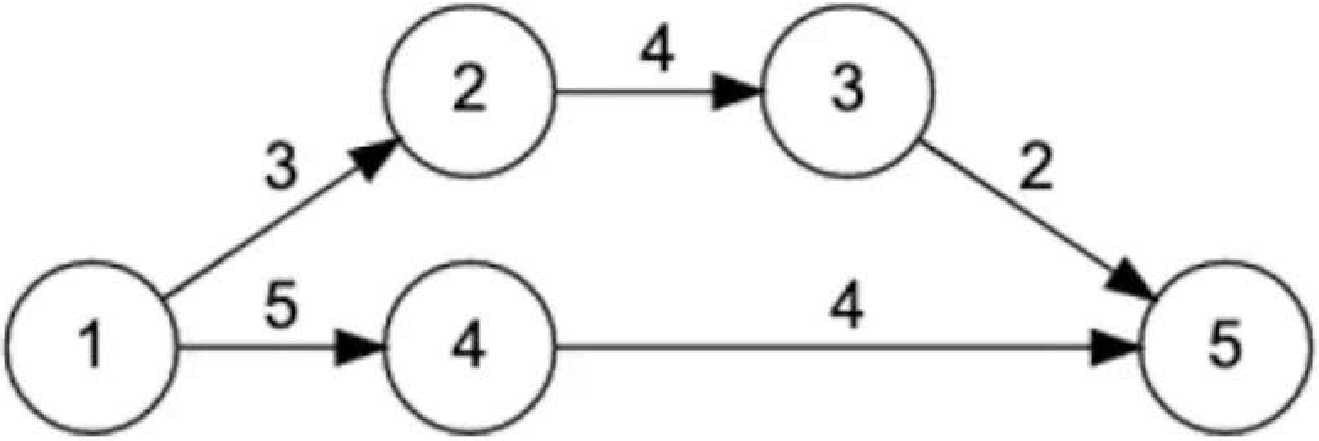


Figure: Sample Input 1

Here the valid travel logs are $\{0, 3, 7, 9\}$ and $\{0, 5, 9\}$.

Solution Sketch, Part 1

- Run *Dijkstra's Algorithm* to find the shortest path from vertex 1 to all other vertices.
- From here on, let dist(u) be the minimum amount of time needed to travel from vertex 1 to vertex u.
- A directed edge (u, v, w) can be on some shortest path leaving vertex 1 if and only if $dist(u) + w = dist(v)$.
- However, it still may not be valid for our purposes if the edge disagrees with the TraveLog.

Valid edges

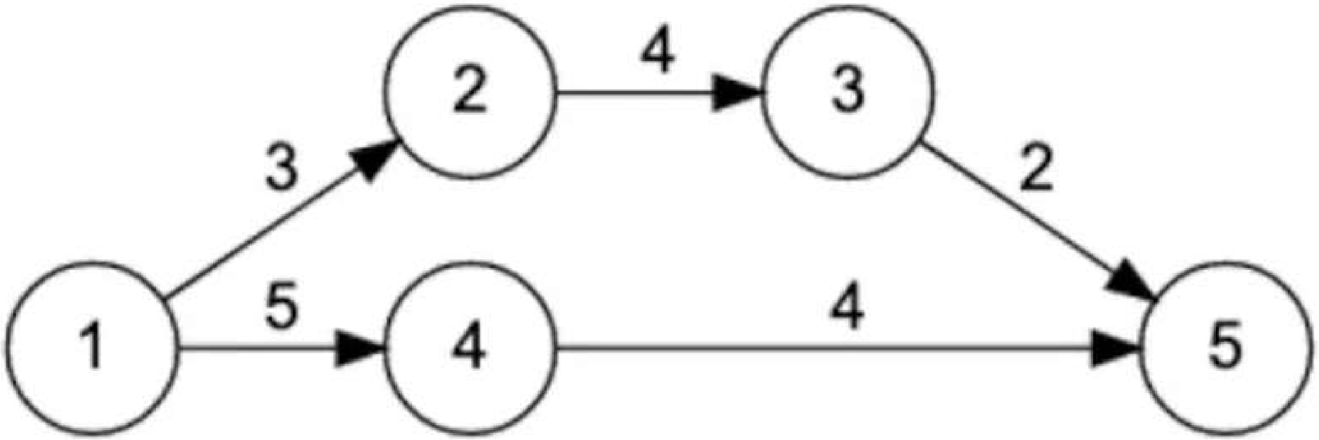


Figure: Sample Input 1

- If 3 is in the TraveLog, why does that invalidate the edge $(1, 4, 5)$?
- This "skips over" a vertex that should exist in our path.
- An edge is valid if and only if $\text{dist}(u) + w = \text{dist}(v)$ and there are no entries in the TraveLog in the interval $(\text{dist}(u), \text{dist}(v))$.

Solution Sketch, Part 2

- By only considering valid edges, we have a *DAG*.
- Counting the number of paths in a DAG is a classic dynamic programming problem that can be solved in $\mathcal{O}(n+m)$.time.
- Let ways(u) be the number of paths from vertex 1 to vertex u. Initially, ways(1) $= 1$ and ways(u) $= 0$ for all other vertices.
- Process the vertices in increasing order of dist(u), and for each edge (u, v), increase ways(v) by ways(u).

Implementation notes

- We cannot scan the whole TraveLog for each edge, so have to be more efficient when checking edge validity.
- Sort the TraveLog, then either binary search for each edge or walk through the TraveLog in lockstep with the dynamic programming algorithm.
- The number of paths between two vertices can be exponential, only store counts of 0, 1, or > 1.
- If the TraveLog has distances greater than dist(n), then there is no solution. Add a check at the end for this case.
- Overall runtime: $\mathcal{O}(m \log n + d \log d + n)$. You might have an extra $\mathcal{O}(m \log d)$ factor depending on implementation.

icpc.foundation

icpc international collegiate programming contest

ICPC North America Championship 2021

Mountainous Palindromic Subarray

The Problem

- Given an array of integers, find the length of the longest subarray that is a palindrome, and which rises then falls.

- For example: in **`2 1 2 3 2 1 7 8`**
- The longest MPS is **`1 2 3 2 1`**, of length **`5`**

The Solution

```
// This is the default if we find no MPS's
int longest = -1;

// Look for the longest, starting at each point (except the ends, obviously)
for( int i=1; i<n-1; i++ )
{
    // Count the number of steps we can take, where at each step:
    // - We're not off of the array
    // - Each step (in both directions) is smaller than the previous (Mountainous)
    // - The upper and lower steps are equal (Palindromic)
    int offset = 1;
    for(;;)
    {
        int ihi = i+offset;
        int ilo = i-offset;
        if( ihi>=n || ilo<0 || x[ihi]!=x[ilo] || x[ilo]>=x[ilo+1]  ) break;
        ++offset;
    }

    // Must be at least 3 to be Mountainous
    int count = 2*offset - 1;
    if( count>2 && count>longest ) longest = count;
}

ps.println( longest );
```